

Da codice etico unisa <http://web.unisa.it/uploads/rescue/41/76/codice-etico-e-di-comportamento-unisa.pdf>

ART. 43 – VIOLAZIONE DEI DOVERI DEL CODICE - STUDENTI

1. La violazione delle norme del presente Codice da parte degli studenti può dar luogo a sanzioni disciplinari, ai sensi del Regolamento Studenti dell'Ateneo.
2. Quando siano accertate attività tese a modificare indebitamente l'esito delle prove o impedirne una corretta valutazione, il docente o altro preposto al controllo dispone l'annullamento delle prove medesime e la segnalazione al Rettore ai fini dell'attivazione del procedimento disciplinare ai sensi del Regolamento studenti.

Da Regolamento studenti unisa http://web.unisa.it/uploads/rescue/31/19/reg_studenti_2014_web.pdf

ART. 40 – SANZIONI DISCIPLINARI A CARICO DEGLI STUDENTI

1. Le sanzioni che si possono comminare sono le seguenti:

- a) ammonizione;
- b) interdizione temporanea da uno o più attività formative;
- c) esclusione da uno o più esami o altra forma di verifica di profitto per un periodo fino a sei mesi;
- d) sospensione temporanea dall'Università con conseguente perdita delle sessioni di esame.

2. La relativa competenza è attribuita al Senato accademico, fatto salvo il diritto dello studente destinatario del provvedimento di essere ascoltato.

3. L'applicazione delle sanzioni disciplinari deve rispondere a criteri di ragionevolezza ed equità, avuto riguardo alla natura della violazione, allo svolgimento dei fatti e alla valutazione degli elementi di prova. Le sanzioni sono comminate in ordine di gradualità secondo la gravità dei fatti.

4. La sanzione è comminata con decreto rettorale.

5. Tutte le sanzioni disciplinari sono registrate nella carriera scolastica dello studente e vengono conseguentemente trascritte nei fogli di congedo.

Esercizio 1. Sviluppare, completando il progetto fornito (Esercizio1), una funzione (che può invocare ulteriori funzioni) "int isBalanced(char *exp)" per verificare se una data espressione aritmetica è ben bilanciata rispetto a tre tipi di parentesi: (), [], {}

$(4 + a) * \{[1 - (2/x)] * (8 - a)\}$ è ben bilanciata

$[x - (4y + 3) * (1 - x)]$ non è ben bilanciata

N.B.: per semplicità supponiamo che non esista un ordine di priorità fra i tre tipi di parentesi:

$(a + \{b - 1\}) / [b + 2]$ è ammessa come valida

ATTENZIONE: affinché l'esercizio venga valutato è **necessario** che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.

Il main deve testare la funzione sulle seguenti espressioni (**NON chiedere input all'utente da tastiera, inserire tutto nel codice C**):

1. $(1+[x+\{5+4\}-7])$

2. $(1+2$

3. $(2+[3*4+\{5\})$

4.

5. $[$

L'output del programma deve essere come segue

1) Valuto: $(1+[x+\{5+4\}-7])$

L' espressione e' bilanciata

2) Valuto: $(1+2$

L'espressione non e' bilanciata

3) Valuto: $(2+[3*4+\{5\})$

Programmazione e Strutture Dati - Prova Pratica

L'espressione non e' bilanciata

4) Valuto:

L' espressione e' bilanciata

5) Valuto: [

L'espressione non e' bilanciata

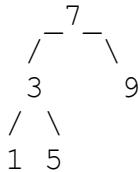
ATTENZIONE: le stringhe non devono essere richieste all'utente ma già definite nel main.

ATTENZIONE: Lavorare esclusivamente nel file **main.c** e **makefile**. Altri file non saranno considerati durante la correzione.

Suggerimento: usare uno stack nella progettazione dell'algoritmo.

Esercizio 2. Si implementi, completando il progetto fornito (Esercizio2), una funzione `int* sumByLevel(BTree)` che prenda in input un albero binario e restituisca un array contenente per ogni livello dell'albero un elemento che contiene la somma dei nodi di tale livello. Utilizzare una funzione `int height(BTree)` che calcoli l'altezza dell'albero. L'albero contiene numeri interi. **Modificare esclusivamente il file main.c** (completando le funzioni già presenti ed eventualmente aggiungendone altre se necessario) **ed il makefile, senza aggiungere o modificare altri file. NON toccare BTree.** Testare nel main la procedura sui seguenti alberi (**NON chiedere input all'utente da tastiera, inserire tutto nel codice C**):

- Un albero vuoto;
- Un albero consistente della sola radice contenente il numero 2;
- Un albero istanziato come di seguito



- Un albero casuale con 7 nodi

Utilizzare la funzione `newRandomTree(int nNodes)` per creare l'albero casuale (fornita dall'ADT) e la funzione `printHeightAndSumByLevel(BTree)` per stampare altezza e somma per livelli dell'albero.

ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.

Esempio di output: (**NOTA: il vostro programma DEVE riprodurre questo output, eccetto per l'albero causale che dovrà essere differente ad ogni esecuzione del programma**)

Albero:

Altezza albero: 0

Somma nodi per ogni livello: 0

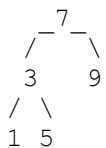
Albero:

2

Altezza albero: 0

Somma nodi per ogni livello: 2

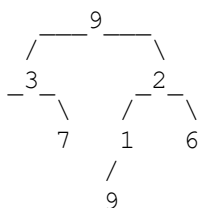
Albero:



Altezza albero: 2

Somma nodi per ogni livello: 7 12 6

Albero:



Altezza albero: 3

Somma nodi per ogni livello: 9 5 14 9

Esercizio 3. Si implementi, completando il progetto fornito (Esercizio3), una procedura ricorsiva che inverte il contenuto di una coda, lasciando **solo gli elementi inferiori ad un dato elemento fornito come parametro** (vedere esempio di output). **Modificare esclusivamente il main.c** aggiungendo le funzioni necessarie e completando `int main()`. Utilizzare i soli operatori di base della coda, **senza utilizzare alcuna struttura dati ausiliaria**.

Testare la procedura sui seguenti **parametro/coda** leggendo l'input da file `input.txt` (fornito) dove **ogni riga corrisponde al parametro seguito dal contenuto di una lista** (**NON chiedere input all'utente da tastiera**):

- Parametro: "ciao"; Coda: coda_vuota
- Parametro: "ciao"; Coda: "ciao"
- Parametro: "ciao"; Coda: "ciao", "bella", "gente", "come", "va"
- Parametro: "dado"; Coda: "ciao", "bella", "gente", "come", "va"
- Parametro: item_casuale; Coda: 7 item causali
(per generare gli item casuali usare `randomItem()`). **NB: nel file input.txt l'elemento casuale corrisponde a !. Quindi quando si trova tale elemento va inserito nella lista un elemento casuale con `randomItem()`.**

ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.

Esempio di output: **(NOTA: il vostro programma DEVE riprodurre questo output, eccetto per la coda causale che dovrà essere differente ad ogni esecuzione del programma)**

```
Elemento parametro: ciao
Coda originale:
Coda invertita:
```

```
Elemento parametro: ciao
Coda originale: ciao
Coda invertita:
```

```
Elemento parametro: ciao
Coda originale: ciao   bella gente come   va
Coda invertita: bella
```

```
Elemento parametro: dado
Coda originale: ciao   bella gente come   va
Coda invertita: come   bella ciao
```

```
Elemento parametro: cyv
Coda originale: rcr   bs   kk   wtn   azj   yop   azd
Coda invertita: azd   azj   bs
```